链滴

Python-CookBook：34、给字符串中的变量名做插值处理

作者：zhaolixiang

原文链接：https://ld246.com/article/1566354881352

来源网站：链滴

问题

我们想创建一个字符串，其中嵌入的变量名称会以变量的字符串值形式替换掉。

解决方案

Python并不直接支持在字符串中对变量做简单的值替换。但是，这个功能可以通过字符串的format()法近似模拟出来。示例如下：

```
s='{name} has {n} messages.'
print(s.format(name='Mark',n=10))
```

输出

```
Mark has 10 messages.
```

另一种方式是，如果要被替换的值确实能在变量中找到，则可以将format_map()和vars()联合起来使，示例如下：

```
s='{name} has {n} messages.'
name='mark'
n=10

print(s.format_map(vars()))
print(vars())
```

输出：

```
mark has 10 messages.
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <_frozen_importlib_external.SourceFileLoader object at 0x1048804a8>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__': '/Volumes/UP/python_cookbook/34-2.py', '__cached__': None, 's': '{name} has {n} messages.', 'name': 'mark', 'n': 10}
```

有关vars()的一个微妙的特性是它也能作用于类实例上。比如：

```
s='{name} has {n} messages.'

class Info:
    def __init__(self,name,n):
        self.name=name
        self.n=n

a=Info('Mark',10)
print(s.format_map(vars(a)))
print(vars(a))
```

输出：

```
Mark has 10 messages.
{'name': 'Mark', 'n': 10}
```

而format()和format_map()的一个缺点则是没法优雅地处理缺少某个值的情况。例如：

```
>>> s.format(name='Guido')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'n'
>>>
```

避免出现这种情况的一种方法就是单独定义一个带有__missing__()方法的字典类，示例如下：

```
s='{name} has {n} messages.'
name='mark'
class safesub(dict):
    def __missing__(self, key):
        return '{'+key+'}'
print(s.format_map(safesub(vars())))
```

输出：

```
mark has {n} messages.
```

如果发现自己在代码中常常需要执行这些步骤，则可以将替换变量的过程隐藏在一个小型的功能函数，这里要采用一种称之为“frame hack”的技巧。示例如下：

```
import sys

class safesub(dict):
    def __missing__(self, key):
        return '{'+key+'}'

def sub(text):
    return text.format_map(safesub(sys._getframe(1).f_locals))

name='mark'
n=10
print(sub('Hello {name}'))
print(sub('You have {n} messages'))
print(sub('You favorite color is {color}'))
```

输出：

```
Hello mark
You have 10 messages
You favorite color is {color}
```

2.15.3 讨论

多年来，由于Python缺乏真正的变量插值功能，由此产生了各种解决方案。作为本节中已给出的解方案的替代，有时候我们会看到类似下面代码中的字符串格式化操作：

```
>>> name = 'Guido'
>>> n = 37
>>> '%(name) has %(n) messages.' % vars()
```

```
'Guido has 37 messages.'
>>>
```

我们可能还会看到模板字符串（template string）的使用：

```
>>> import string
>>> s = string.Template('$name has $n messages.')
>>> s.substitute(vars())
'Guido has 37 messages.'
>>>
```

但是，format()和format_map()方法比上面这些替代方案都要更加现代化，我们应该将其作为首选。用format()的一个好处是可以同时得到所有关于字符串格式化方面的功能（对齐、填充、数值格式化），而这些功能在字符串模板对象上是不可能做到的。

在本节的部分内容中还提到了一些有趣的高级特性。字典类中鲜为人知的__missing__()方法可用来处缺少值时的行为。在safesub类中，我们将该方法定义为将缺失的值以占位符的形式返回，因此这里会抛出KeyError异常，缺少的那个值会出现在最后生成的字符串中（可能对调试有些帮助）。

sub()函数使用了sys._getframe(1)来返回调用方的栈帧。通过访问属性f_locals来得到局部变量。无赘言，在大部分的代码中都应该避免去和栈帧打交道，但是对于类似完成字符串替换功能的函数来说这会是有用的。插一句题外话，值得指出的是f_locals是一个字典，它完成对调用函数中局部变量的贝。尽管可以修改f_locals的内容，可是修改后并不会产生任何持续性的效果。因此，尽管访问不同栈帧可能看起来是很邪恶的，但是想意外地覆盖或修改调用方的本地环境也是不可能的。